NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

A CHECKER-PLAYING PROGRAM

by

De Ford Eugene Cochran

December 1971

71

Unclassified
Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Naval Postgraduate School Monterey, California 93940	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

3. REPORT TITLE

A Checker-Playing Program

4. DESCRIPTIVE NOTES (Type of report and, inclusive dates)
Master's Thesis; December 1971

5. AUTHOR(S) (First name, middle initial, last name)

De Ford Eugene Cochran

6. REPORT DATE December 1971	7a. TOTAL NO. OF PAGES 71	7b. NO. OF REFS 9
8a. CONTRACT OR GRANT NO. b. PROJECT NO. c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	

10. DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Naval Postgraduate School Monterey, California 93940

13. ABSTRACT

This paper describes the design of a computer program which plays checkers. The program's objective was to play a respectable game without using any rote memory and with a minimum amount of look-ahead, by relying upon static evaluations of various features of the checker-board. An historical background of computer game-playing is presented with detailed explanation of the important concepts as they appeared in the literature. The techniques employed in this program are explained and compared to those used in previously written programs. Major program processes are diagrammed and documented games played by the program appear in the appendices.

DD FORM 1 NOV 65 1473 (PAGE 1)
S/N 0101-807-6811

Unclassified
Security Classification

A-31408

Unclassified
Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Checkers						
Computer Game Playing						

DD FORM 1 NOV 65 1473 (BACK)
S/N 0101-807-6871

Unclassified
Security Classification

A-31409

A Checker-Playing Program

by

De Ford Eugene Cochran
Lieutenant, United States Navy
B.A., University of Washington, 1967

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1971

Author De Ford Eugene Cochran

Approved by: ______________________
Thesis Advisor

[illegible]
Chairman, Department of Mathematics

[illegible]
Academic Dean

ABSTRACT

This paper describes the design of a computer program which plays checkers. The program's objective was to play a respectable game without using any rote memory and with a minimum amount of look-ahead, by relying upon static evaluations of various features of the checkerboard. An historical background of computer game-playing is presented with detailed explanation of the important concepts as they appeared in the literature. The techniques employed in this program are explained and compared to those used in previously written programs. Major program processes are diagrammed and documented games played by the program appear in the appendices.

TABLE OF CONTENTS

LIST OF DRAWINGS

I. INTRODUCTION

Artificial intelligence is the science of making computers do things that would require intelligence if done by men. One of the first areas of interest of artificial intelligence research was computer game-playing. There were several reasons for this interest.

Games provide a direct contest between man and machine in a well defined problem environment. The problem environment is sufficiently complex to require intelligence and reasoning by human game players. In addition, games are not difficult to represent in computer programs.

Board games, such as checkers, and especially chess, have long histories. These games are governed by rules which define a problem environment. The goal of the game, to win, is clear. The rules also define the legal moves. Each game is a sequence of moves and responses. Because of the extreme complexity of the structure of all possible moves, humans must use some form of strategy, or general plan, to help to simplify the process of selecting moves. Even with the many strategies that have been devised by men, no complete understanding of chess exists.

In theory, chess is a finite game. There are a finite number of board positions which are connected to one another by a network of legal moves. From any board position, a finite number of moves is available. The game can be represented by a tree, with the board positions as nodes, and possible moves as branches. This tree is finite, but so large that it is not feasible to attempt to explore all possible moves from the beginning to end. At each node in the move tree a decision must be made with only a limited amount of information available. The decision-making process required at each position involves evaluation of alternate moves.

This complex but well defined problem has been programmed for play on computers. Programs using chess knowledge and careful analysis of possibilities exist which are successful to the point that games have been won in tournament play.

II. COMPUTER GAME PLAYING

Prior work in computer game playing has resulted in the development and analysis of many important concepts. Investigation of this literature is necessary before attempting to produce a new game-playing program. Many elements found in previously written papers and programs are common to almost all successful game-playing programs. Because of the influence this body of literature has had on the present work, a brief description of some of the major prior work will be presented.

A. SHANNON'S PROPOSAL FOR CHESS

In 1949, Claude Shannon published a paper which discussed some of the problems involved in writing a chess-playing program [Ref. 1]. His basic outline has been used in virtually all attempts to produce computer chess-playing programs. He proposed the following framework:

1. Consider all possible moves in the current board position.

2. Analyze each move to obtain a measurement of the value of the move.

3. Select the best move on the basis of the values.

In considering a particular move, the program would explore all responses available to the opponent. The program would consider each move available after the opponent's response, and so on. This process of exploring continuations is known as look-ahead.

Shannon's proposal was to explore continuations to a fixed depth in the move tree. Because the tree is so large, this exploration cannot be expected to reach terminal nodes. After looking-ahead to a given depth, each of the board positions reached was evaluated.

The result of evaluation of the board position was a numerical measure. This measure was calculated by summing weighted factors - each factor corresponding to a feature of the chess board that chess experts considered important. Each factor was computed for the particular board position and then multiplied by a weight, which represented the relative importance of that factor. The sum of these weighted factors was then assigned to the board position as its value.

With the look-ahead complete and the resulting boards evaluated, Shannon then used the board values to derive a value for each of the original possible moves. The procedure he used to work back up the tree was called minimaxing. The minimaxing process assumes that at each node where the program is to move, it selects the move with the maximum value and at each node where the opponent is to move, the opponent will select the move with the minimum value. This process starts at the terminal nodes, which are board positions with values, and works backwards up the tree to the current board position. When all of the nodes have been assigned a value, the program selects the move leading to the node with the largest value.

A simple example will help to clarify this procedure. Figure 1 shows a situation where black is to move. Black has four choices, and for purposes of this example, look-ahead is only conducted to explore white's immediate response.

After each of black's moves, white has two possible replies. These eight boards have been evaluated as indicated. At each of nodes (1), (2), (3), and (4), black assumes that white will choose the move resulting in the node with minimum value. Hence, black assumes that node (1) has a value of minus one, node (2) a value of minus one, node (3) a value of zero, and node (4) a value of one. Black then selects move four, since he is assured a value of at least one.

Shannon proposed that his program could play better chess as it was able to look farther ahead in the move tree. Minimaxing was intended to insure that the alternate moves were considered in light of the most likely board position value.

B. TURING'S CHESS PROGRAM

A. M. Turing produced a chess-playing program in 1950 which embodied many of Shannon's proposals [Ref. 2]. All legal moves were considered at each node. Boards were evaluated after look-ahead, and the preferred move was selected using minimaxing.

Turing introduced the notion of a dead board position. A position is dead if no moves exist which will drastically change the evaluation of the position.

In order to reduce the magnitude of the computations involved in look-ahead, Turing only evaluated board positions that were dead. For example, if the look-ahead terminated in the middle of an exchange of pieces, the continuations were extended until the exchange was complete. At this point, it made sense to compute the total value of each player's pieces and to compare them.

The value of material was considered dominant in the evaluation of these dead positions. If more than one branch of the tree led to the same material balance, additional factors were considered in making the choice between these moves. Various factors, such as mobility, control of the center of the board, etc., were weighted and summed to provide the additional evaluation.

The program was hand-simulated and its only published game was weak. However, Turing did produce the first real program for chess playing, and led the way for further developments.

C. BERNSTEIN'S CHESS PROGRAM

During the late 1950's, Alex Bernstein wrote a chess playing program which was run on a computer [Ref. 3]. Bernstein's program used techniques developed previously; but rather than considering all legal moves, the program only considered a fraction of the possible legal moves at any board position.

The program contained routines called plausible move generators which selected a number of moves from among all legal moves. The object of the move generators was to find moves which somehow improved the program's board position. The program was sensitive to several relationships, or board features, which chess experts considered important in move selection. Examples of these features are king safety, control of the center of the board, defense of men, etc. The move generators were executed, one at a time, until a maximum of seven plausible moves was produced. The order of execution of the generators was based on a priority scheme: king safety first, etc.

This selectivity allowed the program to do a more detailed analysis and evaluation of the moves. However, the program did overlook a good move, on occasion, because no move generator proposed it. Move generators were designed to generate moves based on information available from expert chess players, and additional generators could be added if needed.

The program looked ahead two moves. At each node in the look-ahead, the plausible move generators were executed. Hence, at each node, a maximum of seven moves was explored. A board was evaluated by comparing two weighted sums - one for red and one for black. The terms of the sums were material, area control, king defense, and mobility. Shannon's minimaxing procedure was then used to select the best alternative.

Bernstein's program played a passable game against amateurs. The program played stronger in the beginning and middle game than in the end game. He proposed adding new plausible move generators and changing some of the decision making routines to improve the weak end-game play. None of the games played by the program was found in the literature.

D. NEWELL, SIMON, AND SHAW'S CHESS PLAYER

A. Newell, H. A. Simon, and J. C. Shaw described the NSS Chess Player in a paper published in 1958 [Ref. 4]. The NSS Chess Player extended some of the concepts presented above. The program was intended to describe and help understand human thinking and decision-making processes. Because of this emphasis, the NSS Chess Player did not evaluate moves using the weighted sums, as was done in earlier programs. The authors did not believe this form of evaluation was present in human decision making. The humanlike problem-solving methods employed put the program in the area of artificial intelligence concerned with modelling human problem-solving methods.

The program was written in four language levels. The first and second levels were machine code and IPL IV[1] code, respectively. The third level was a basic chess vocabulary. The elements of this vocabulary expressed concepts of the chess game. An IPL program which measured or tested for the particular concept was associated with each element of the vocabulary. These routines responded to particular situations, such as men being on the same diagonal, or men bearing on a certain square, etc. Using this vocabulary of approximately 100 routines, the chess-playing program itself was written.

The program began play by determining the state of the game. For the purposes of the program, the four states of the game were beginning, early-middle, late-middle, and end. The state was derived by analyzing various features of the present board situation. After the state had been determined, a set of goals, appropriate to the situation was selected. These goals were terms associated with game, such as center control, king safety, mobility, and the like.

The program used the state of the game to determine the priorities among the goals. The priorities were reflected in the order of the goals in the goal list.

Associated with each goal was a move generator. The move generator, as in Bernstein's program, generated moves which would improve one aspect of the board. As an example, the material balance generator would generate a move that eliminates a threat to the program's pieces.

[1]IPL IV is a list processing language; one of a series of languages developed by Newell, et al.

For each goal, there existed an analysis routine which did the look-ahead. The resulting board was reduced to a dead position, if necessary, and then several evaluations were performed. The board was evaluated with respect to each goal and given a numerical value for that goal. The total evaluation resulted in a vector of values, one for each of the goals. The values in the vector indicated the acceptability or unacceptability of the move for each specific goal.

The NSS Chess Player applied Turing's dead position concept in a more general fashion than did Turing. Turing reasoned that evaluations which occurred in the middle of an exchange of pieces did not reflect a true measure of material. Similarly, Newell, Simon, and Shaw reasoned that a board position was dead with respect to a given feature if no plausible moves would result in a drastic change in the feature. If the board position was not dead with respect to the feature, further continuations were explored. Using this generalization of the dead position concept, the NSS program was designed to evaluate a board only if the board position was dead with respect to all the board features associated with the goals. When a board position was reached which was dead with respect to all the features, the evaluation of the board was performed.

In order to minimax and to make a final choice of moves, the vectors associated with each dead position had to be compared. The values in the evaluation vector were in the same order as the goals in the goal list. To compare two vectors, the first components were compared. The node with the largest first component was selected. If two nodes had the same value, the second value determined the choice, and so on.

This selection process gave overwhelming importance to the first goal on the goal list. Great care was taken to insure that the goals selected for each state of the game were relevant to the particular situation and emphasis was also placed on proper ordering of the goals.

The move selected by the program at a node was normally the first move to meet or exceed an acceptance level. The acceptance level was determined in the goal selection analysis. If none of the generated moves produced a result which reached the acceptance level, the move with the best evaluation was selected. In the normal case, only a fraction of the generated moves were evaluated. Only when none of the moves reached the acceptance level, did the program have to evaluate each move.

E. THE GREENBLATT CHESS PROGRAM

The Greenblatt Chess Program used plausible move generators to limit the width of the move tree, as did Bernstein. Continuations were explored to a certain depth and then minimaxing was used to evaluate the alternatives. This program had a great deal of chess know-how built into it. It contained a table of opening positions and selected replies to help it avoid well-known traps. The designers of the program are expert chess players themselves, and they have programmed in much of their own knowledge of the game.

The program used an algorithm for discarding some branches of the move tree, known as alpha-beta tree pruning. Application of this algorithm eliminated many of the static evaluations required for standard minimaxing. For an example of alpha-beta tree pruning, consider figure 2. Figure 2 represents part of a move tree. At the root node, black is to move. Black first explores moving to node (1). The continuations are explored, producing nodes (3), (4), (5), and (9) through (17). Static evaluations are performed on nodes (9), (10), and (11). Minimaxing yields a value of five for node (3). Node (12) is evaluated as a nine. Since red will choose to minimize the evaluation of node (1), and black will seek to maximize the evaluation of node (4), nodes (13) and (14) need not be evaluated. Red will move to node (3) rather than node (4), since a move to node (4) allows black to gain more advantage than does node (3). Node (5) evaluates to one. Hence, node (1) is assigned a value of one. Similarly, nodes (8), (24), (25), and (26) need not be evaluated. Node (7) yields an evaluation which will cause node (2) to be an illogical choice for black. If black selects node (2), red can cause a final evaluation of, at most, minus one for the move. This algorithm greatly reduced the amount of calculation required in move selection since only a fraction of the nodes at each level must be evaluated.

The plausible move generators were designed to assign the highest priority to the best move. The alpha-beta pruning was most effective in this case. The alpha-beta pruning can reduce the search workload by as much as a factor of 100. To speed the search through the tree even more, a list of boards already considered was kept, to preclude evaluating the same board twice.

Greenblatt's program performed well in play against humans. The program has played hundreds of complete games and has won some games in tournament competition. The program beats about 80 per cent of its non-tournament opponents and in the April 1967 Massachusetts amateur tournament, it won the Class D trophy.

The game-playing programs discussed above have all been aimed at playing the game better by improved implementation of program logic. All have relied heavily upon Shannon's original proposal.

F. SAMUEL'S CHECKER PLAYER

A. L. Samuel's checker-playing program was designed to improve its performance with experience. The program used many of the techniques discussed above in the chess programs. It looked ahead a few moves and evaluated dead positions. The look-ahead often extended to ten complete moves. This level of look-ahead allowed the program to conduct an extremely detailed analysis of available moves. The success of Samuel's program is a result of this detailed analysis.

The evaluation of the moves was accomplished by calculating the value of a linear polynomial. The terms of the polynomial were measurements of various features of the checkerboard. As in chess, these features include center control, mobility, material balance, etc. The coefficients of the terms reflected the relative importance of the corresponding features in the overall evaluation. Minimaxing was used to evaluate possible moves. The move with the highest polynomial value was selected by the program.

Samuel identified two forms of learning in his program: rote learning and generalization. The rote learning technique involved storing board positions and their evaluations. If a stored board was subsequently encountered in actual play or look-ahead, the program could look-up the evaluation of the board, rather than calculate it. This form of learning, although not very advanced, allowed the program to look much farther ahead than might otherwise be possible. If, for example, look-ahead was only extended to two moves, consider the case in which the program recognized a board it encountered at the dead position. The stored evaluation of this board could very well have been based on two move look-ahead itself. Hence, the evaluation used for minimaxing could actually be based on four move look-ahead. Various techniques were used for cataloguing boards, detecting redundancies, and discarding boards.

Learning by generalization involved changing the terms and coefficients of the polynomial. The terms of the polynomial were obtained by measuring features of the board position. There was a total of 38 features available to the program. Of these, only 16 were used in the polynomial at any one time. The remaining terms were kept on a reserve list.

The generalization learning technique allowed the program to change coefficients in the polynomial when the values produced by the polynomial were not considered to accurately reflect the value of the board. The polynomial was tested by first applying it to each board position encountered in actual play and saving the value obtained. Look-ahead was then performed to a given depth. Using minimaxing, the program computed a backed-up evaluation of the board position. This backed-up value was compared to the value computed without the look-ahead. The value obtained using look-ahead was considered to be a more accurate measure of the board position's value. If the initial value was greater than the backed-up value, the positive coefficients in the polynomial were reduced and the negative coefficients were increased. If the initial value was less than the backed-up value, converse action was taken.

After each move, the program determined the term in the polynomial with the lowest coefficient. This term had the least effect on the value of the polynomial. If a particular term occupied this position for eight moves, the term was transferred to the bottom of the reserve list. The term at the head of the reserve list was transferred into the polynomial and given a coefficient of zero. The coefficient was then altered during play as described above. This learning technique provided the program with the ability to adapt to the play of each opponent.

The rote learning technique provided improvement in the beginning and end-game play. The generalization technique improved middle-game play. The performance of Samuel's checker player indicated that learning techniques could be programmed to help the computer learn to play games better.

III. ORIGIN OF THIS WORK

This work is a continuation and extension of a class project begun in July, 1971. The project was undertaken for an advanced topics in Computer Science course taught at the Naval Postgraduate School, Monterey, California. Thirteen students participated in the project. The goal of the project was to produce a program to play an interesting game of checkers against a human opponent. The design of the program was to draw heavily upon work done previously in the field of computer game-playing. In addition to consulting the literature on computer game-playing, some initial research into the literature dealing with the game of checkers itself was required to acquaint the designers with terms used in describing and playing the game.

A. THE BASIC PROGRAM PRODUCED BY THE CLASS PROJECT

The program consisted of a basic checkers vocabulary, similar to the vocabulary used in the NSS Chess Player, and playing routines. The vocabulary routines measured various features of a board position, such as mobility, and provided the basic machinery needed to represent the board. This basic machinery manipulated the board to reflect moves or jumps.

The playing routines used the vocabulary to select a subset of all legal moves for consideration, to reduce the alternatives to dead positions, to evaluate the dead positions, and to select a move. The program was goal oriented in the same sense that the NSS Chess Player was goal oriented. The first step in program flow was to select and order a set of goals. These goals were associated with important features of the board such as mobility, defense of men, etc. Associated with each goal was a move generator which selected moves to improve the board feature associated with the goal.

The program did not have the ability to learn. Any improvement in performance was the result of additional programming.

Each static board was evaluated with respect to material condition, center control, and mobility. A vector containing these three elements, plus a weighted sum of the three elements, constituted the final evaluation.

1. Goal Selection and Ordering

The first step in goal selection and ordering was to determine in which of four stages of the game the program was playing. The number of pieces on the board determined the stage of the game as follows:

STAGE	NUMBER OF PIECES
Beginning	19 to 24
Early middle	13 to 18
Late middle	7 to 12
End	1 to 6

For each stage, a relative importance, or weight, was assigned to the board features of mobility, center control, and material balance. These weights were used to compute a weighted sum of relative mobility, center control, and material balance, called advantage. This value was then used to select a list of goals. The order in which the goals appeared in the list reflected the relative priorities assigned to the goals. The value of advantage was significant in this selection process since it indicated the relative strength of the program's position. If, for example, the value of advantage was low in the beginning stage, a list of goals which gave high priority to gaining control of the center would be selected.

Each of the four goal lists was a permutation of the four goals: MOBILITY, CENTER-CONTROL, GET-KING, and CAPTURE. The priorities were assigned to the goals based on information obtained from published literature on checkers. The stage of the game and the overall evaluation of the program's situation were used to determine the selection of the goal priorities.

In addition to the goal list, the program also saved the value of advantage and the three weighted board features used in computing advantage. These four values constituted the vector evaluation of the present board. This vector was used later for evaluation of moves. Positive values indicated black dominance, negative values indicated red dominance, and zeros indicated an even balance in the features associated with each of the vector elements.

After the basic goal list was selected, additional goals were added to the list which allowed the program to test for and respond to particular situations. The goals SHOT and KIL were always added to the goal list. The move generator associated with SHOT searched the board for a specific pattern of pieces. If this pattern was present, the program could move to sacrifice one piece in exchange for the opponent losing two. This advantageous exchange is known as a two-for-one-shot. The move generator for the KIL goal recognized patterns which allowed the program to capture an enemy piece which was not well defended.

If the board analysis during goal selection determined that a threat to one of the program's pieces existed, the goal BLOCK was placed on the goal list. The move generator associated with BLOCK searched for moves to eliminate the threat. If a jump was available to the program, the goal JUMP was given highest priority on the goal list because the rules of checkers require that available jumps be taken.

The list of six to eight goals and the vector evaluation of the present board were used in the move generation phase.

2. Move Generation

Each goal (except JUMP) had a move generator associated with it. These move generators investigated some subset of all legal moves and selected the move which resulted in the most improvement in the appropriate board feature.

The move generators investigated moves by making the move, reducing the board to a dead position, and then analyzing the resulting board position. When the dead position had been evaluated, the move generator compared this evaluation with the evaluation of the present board. The generator saved the best board evaluation and the move executed to reach that board. When each move evaluation was compared to the present board evaluation, the move was rejected if material was lost, other than in even exchanges, or if the mobility or center control features were drastically degraded by the move. If none of the moves investigated improved the board feature associated with the goal, the move generator indicated this fact and the generator for the next highest priority goal was executed.

As soon as one of the move generators found an acceptable move - that is, one that sufficiently improved the goal being considered - the program executed the move.

3. Analysis and Evaluation

Board positions encountered after a proposed move were analyzed to determine if the opponent would now be forced to jump. If this was not the case, the board was considered dead, and the evaluation was performed. If the opponent did have a jump, the jump was executed and the board was analyzed to determine if the program was now forced to jump. This process continued until a node in the tree was reached at which one player had a choice to make. This board was then evaluated to measure the effect of the move.

The evaluation performed on the dead position was accomplished using a section of the goal generation routine. The dead position was evaluated with respect to relative mobility, center control, and material balance. The weighted sum, advantage, was also computed. The various components of the vector were then compared with those of the present board's evaluation.

B. PERFORMANCE OF THE BASIC PROGRAM

The program could play the game as long as moves which were acceptable to one of the move generators were available. The basic machinery for moving, jumping, measuring features of the board, etc., was coded and de-bugged for many situations. Portions of several games were played by students against the program. After a fair beginning game, the program soon lost direction and was easily beaten.

Analysis of the ordering of the goals and the moves selected by the program in these games revealed several areas in which program performance could be improved. The move generators were not sensitive to improvement in material balance. If, for example, the dead position resulting from a move proposed by the mobility move generator did not increase the number of safe moves available to the program, the move was rejected, even if it would result in the capture of enemy pieces. Some of the move generators did not explore all possible means for accomplishing the goal. For example, the move generator for the goal BLOCK only investigated moves to block the jump. It did not consider moving the threatened piece out of danger. If each of the move generators was executed, but none of them found an acceptable move, the program would resign. In many cases, when the program resigned it was not in a completely hopeless situation.

Because the program was a group project, and coding was done by many programmers, there were many inefficiencies in the program coding.

Because of the complexity of the program, the interaction between the basic elements was not always clear. Several bugs existed in the interaction of various routines, which had a drastic effect on the program's performance. For example, the program would occasionally compare the vector evaluations of several moves and make an obviously erroneous selection. The reason for this error lay in the interaction between the move generator and the goal generation routine. As mentioned above, evaluations of dead positions, after a proposed move, were accomplished using the goal generation routines. The calculations performed were based on the stage of the game. If a move resulted in exchanges, sometimes the stage of the game would be changed. The weighting factors used to calculate the elements of the evaluation vector would then change. Hence, direct comparison of vector elements could lead to wrong conclusions and selections.

IV.. GOALS OF THE PRESENT WORK

The basic goal of this work was to write a checker-playing program which decides on the same moves that a human player selects, and for the same reasons. The thrust of the subsequent research and programming was to better implement concepts presented by other authors and to improve the heuristics already employed in the program. Emphasis was also placed on producing a more efficient program flow and locating and resolving logical flaws and omissions.

Another goal of the work was to provide additional goals for move generation. Improvement in the goal definitions, analysis, and the goal selection process was considered vital to improve the program's play. Several analysis routines required additional modifications to cope with situations encountered in play that were not considered in the program. Games were then to be played to test and improve the middle and end-game performance.

Performance of the basic program indicated that a modest amount of look-ahead could greatly reduce the probability of selecting a poor move. The look-ahead would enable the program to detect a move which met all the criteria for selection except that with the resulting board position, the opponent was able to make a move which resulted in a material loss for the program.

To detect such situations prior to move selection, a new routine was written which performed one ply look-ahead. After each of the program's moves was proposed, the routine proposed each of the opponent's replies and reduced the board to a dead position. If material had been lost, other than in an exchange, the routine flagged this move by the program as unacceptable. Using this routine during the analysis phase of move generation also ruled out sacrificing a piece to improve a feature of the board. Because the program did not base move selection on previously selected goals or moves, it was unable to use sacrifices to produce strategic gains in any case. This one ply look-ahead did not restrict the generation of moves, but it did provide a defensive measure to detect some poor moves.

Although the most satisfying method of competition is interactive, the time-sharing system at the Naval Postgraduate School was unable to process the language selected for this program. In early October 1971, an addition to the time-sharing system made interactive play possible. The implementation of this interaction became a secondary goal of the work.

V. CHECKER$_1$

CHECKER$_1$ is a checker-playing program written for use on the IBM 360/67 installed at the Naval Postgraduate School, Monterey, California. The program is written in the list processing language LISP 1.5 [Refs. 7 and 8].

The basic organization of the program is similar to that proposed by Shannon, with the addition of selectivity in move generation, and analysis of dead positions. The program uses goals to help select moves for investigation as did the NSS Chess Player. Numerical evaluations of the present board configuration provide a means to determine what feature of the game should be improved. The program then considers various plausible moves and selects a move based on the goal determined to be most critical. Selection is accomplished by evaluation and analysis of dead positions resulting from proposed legal moves.

Some convention for board representation had to be selected to allow communication between the human player and the program. The standard checkerboard numbering system numbers the squares as shown in Figure 3. A modified numbering system, as in Figure 4, is used in communicating with the machine. This numbering system, as described by Samuel [Ref. 6], numbers the squares from one to thirty-five, omitting nine, eighteen, and twenty-seven. This numbering system is used internally by the program to compute possible moves.

With the modified system, a black man can move to squares numbered four higher, or five higher, than the square it is on. A red man can move to squares numbered four or five lower than the square it is on. Kings can move both ways. If adding four or five to a square results in nine, eighteen, twenty-seven, or a number greater than 35, the new square does not exist. Similarly, if the subtraction yields nine, eighteen, twenty-seven, or a number less than one, the square does not exist.

Modification of the numbering system also facilitates calculation of jumps. A man can only jump into squares minus eight, plus eight, minus ten, or plus ten from the square it is on. The test for existance of a square is the same as for moves. All board square numbers referred to in this paper are from the modified system.

A. BASIC ORGANIZATION OF THE PROGRAM

The program is organized into four steps. First, the program orders goals from the goal set. The ordering is dependent upon the stage of the game and measurements of center control, mobility, and material condition. These goals are independent and goals may be added or removed without affecting the other goals. Plausible moves are then generated, proposed, and evaluated. The evaluation includes analysis of dead positions and measurements of board features. The generation phase either yields an acceptable move or indicates that the program cannot improve the goal, given the present situation. If no move is produced, the program then considers the next goal on the list of goals.

If all the move generators associated with the goals on the goal list fail to produce an acceptable move, the move generation process calls upon two other move generators. The first generator attempts to produce a safe move for the program. If no safe moves exist, the second generator investigates available sacrifices. Men are sacrificed only if the sacrifice will open new avenues toward goal achievement. Otherwise, the program resigns.

B. GOALS

Nine independent goals are presently used. Two other goals are built into the selection process. These two goals are only used if all efforts to achieve the other goals fail.

After determining the stage of the game by counting the total number of pieces on the board, the program assigns a priority to each goal. The priorities are represented by the order in which the goals appear in the goal list and were derived from analysis of published checkers games, played by experts. (Board evaluations used in this analysis were hand-simulations of the program's various evaluation routines.) The ordering of the goals depends on the stage of the game, measurements of various features of the board, and whether or not a jumping situation exists for either player.

In each stage, if the program has at least two possible safe moves, the mobility goal is given low priority. If the program is behind in material balance, the capture goal is given priority, while the exchange goal is not desired.

Throughout the first three of the four stages, center control is given priority if the program does not have at least a slight advantage. The goal associated with advancing men towards kings row increases its priority in each of the first three stages of the game.

In move generation, the program will accept even trades in material to improve the given goal. Since kings have been given twice the value of men, the program will trade two men for a king, or a king for two men.

A brief definition of each goal and the criteria used in ordering the goals follows. The goal generation process is diagrammed in Figure 5.

1. Jump

JUMP is the highest priority goal. If the program finds that it is in a situation which forces a jump, this goal is selected. Further goal generation is terminated.

2. Block

This goal is first on the goal list if the opponent can jump one of the program's pieces. The program first attempts to move another piece, so as to block the jump. If a block is not possible, the program attempts to move the threatened piece to a safe position. If this is not possible, an attempt is made to move another piece to cause the opponent's jump to result in an exchange.

The block is considered the preferred action, since blocking moves consolidate the men nearest to the enemy for mutual support. If, rather than blocking, a piece is moved out of danger, he is frequently isolated from support if future threats arise.

3. Shot

The goal SHOT appears on the goal list to activate a search for certain pre-stored patterns of pieces which allow the program to sacrifice one piece in exchange for capturing two enemy pieces. This move is referred to as a two-for-one shot in checkers literature. The stored patterns do not match all those possible for a two-for-one shot, but only a certain number occurring near the center of the board.

4. Kil

The goal KIL allows the program to notice and capture insufficiently protected pieces which the opponent has advanced near kings row. The move generator looks for a lone enemy man in the program's third row. If there is a man in the third row, and if he can be forced into a situation which results in the program capturing this man, with no exchange resulting, the forcing move is generated.

5. Center Control

Control of the center is considered to be control of squares 15, 16, 20, and 21. A piece contributes to center control if it occupies these squares or the supporting squares (numbers 10, 11, 12, 17, 19, 25, 26). The value of center control is computed as follows. The values of the pieces occupying the center control squares are multiplied by four and summed. The values of the pieces occupying the supporting squares are added to the sum to yield the value of center control. Black men have a value of one, black kings have a value of two; red men a value of minus one, and red kings a value of minus two. A center control value greater than zero indicates a black advantage; zero indicates even control; and less than zero indicates a red advantage.[2]

During the beginning and early-middle stages, the program chooses CENTER-CONTROL as the primary goal when the opponent has the advantage or the control is even. With more than twelve pieces on the board, control of the center is an excellent deterrent to the opponent getting a king. The opponent's mobility is greatly reduced if he does not control the center.

6. Mobility

The overall strategy of checkers is to not allow the opponent to move. Because of this, the program must be concerned with how many safe moves are available to it. Each time the program begins goal generation, it determines its mobility. If less than two safe moves are available, the MOBILITY goal is given top priority. This goal becomes increasingly important as the game moves toward the end.

[2] Appendix C illustrates the square values for computing center control.

7. Get-king

In the middle game, the program attempts to take the offensive by moving a man into kings row. If control of the center and mobility are not critical, the GET-KING goal is given priority. The move-generator for the GET-KING goal scans the board to determine the side on which the opponent appears the weakest. The move which advances a man as close as possible to kings row on the opponent's weak side is considered the best move toward getting a king. A move to the strong side is considered if no move on the weak side advances a man toward kings row. Exchanges which result in a man being moved closer to kings row are acceptable.

8. Exchange

If the program has a material advantage in the late-middle game or end game, it will seek to make exchanges to reduce the game to a winning situation. In the beginning and early-middle game, exchanges are sought to speed the game to later stages. If at any time the program is at a material disadvantage, the EXCHANGE goal is relegated to a position only above sacrificing a man. When a player is behind in material, exchanges degrade his position by moving the game closer to an end-game situation assuring a win for his opponent.

9. Capture

The CAPTURE goal is designed to generate moves which contribute to jumping an enemy piece. The program can accomplish the CAPTURE goal in two ways. The first is to reduce the opponent's mobility. The second is to place one of the opponent's men in a position forcing a move or block to prevent loss of the man. This goal is considered most important in late-middle game and end-game play. Use of this goal in the end game will lead to a win by denying the opponent the ability to move safely. If the program is at a material disadvantage, this goal is given higher priority. Accomplishing this goal can eventually lead to regaining material balance - or even an advantage.

C. MOVE GENERATION

Move generation is accomplished by investigation of various moves which may help improve the goal selected. The move generator for each goal proposes alternative moves and then evaluates the move by measuring features of the resulting dead position. The evaluation determines how the goal was affected and the change in other pertinent board features.

Each move generator follows the same basic outline in generating moves. The four steps are:

1. A subset of all legal moves is selected for investigation.
2. Each selected move is proposed.
3. The resulting board is reduced to a dead position.
4. The dead positions are evaluated and compared.

Methods for selection of plausible moves for consideration vary with the goal involved. The move generator for CENTER-CONTROL only considers moves which improve the value of center control. The move generators for CAPTURE and MOBILITY consider all legal moves. The move generator for GET-KING considers only one side of the board at a time. The move generators for BLOCK, KIL, and SHOT need only look at a small part of the board for appropriate moves.

Once a set of moves has been selected for investigation, each move in the set is proposed. The moves are proposed by actually executing them on the present board configuration. After analysis and evaluation of the new board, the board is restored to its prior configuration.

When a move is proposed, the move generator performs analysis of the board to determine if the board is now a dead, or static, position. A static position is defined as one in which the player who now is to move has a choice to make - in other words, no jumps are forced. If the position is not static, the forced jumps are performed until a static board results. This idea of a static position is identical to Turing's concept of a dead position. Unlike the NSS Chess Player, which considered a generalized version of this concept, the program only considers forced jumps in reducing the board to a dead position.

The algorithm for final choice of a move depends upon which move generator is executed. Each generator has certain criteria for determining if a given move improves the appropriate feature of the board. The final choice is made either by finding the first plausible move that produces an improvement, or by selecting the move that produces the most improvement. Figure 7 compares the move generators and indicates the moves that are generated and how the final choice is made. The move generators are described below.

1. <u>Block</u>

The BLOCK goal requires that no jumps are immediately available to the opponent after the blocking move is executed. Each move which the program can make to block the jump is considered. The first move that does not open another threat is selected.

In the event that no block is possible, the program determines whether the threatened piece has a move available. If any such moves exist, they are proposed and tested as above.

When there is no way to prevent the loss, moves are proposed which place a piece adjacent to the blocking square. These moves usually result in either an exchange or a double jump for the opponent. Each is proposed, and the first move resulting in an exchange is selected.

If there is no way to prevent loss of material, the goal is abandoned, and the next goal processed.

2. <u>Shot and Kil</u>

The move generators for the SHOT and KIL goals search the current board for pre-stored patterns of pieces. The generator for SHOT attempts to match patterns which allow the program to get a two-for-one shot. The generator for KIL searches for patterns in the program's first, second, and third rows which will allow the program to capture an unprotected enemy piece. Each generator produces the move appropriate to the first pattern it recognizes. If none of the pre-stored patterns match on the board, the generators indicate failure, and the next goal on the list is processed.

3. Center Control

When generating moves for center control, after each proposed move, the board is reduced to a static position. The value of center control at this static position is compared with the previous best value. If a move does improve center control, and does not result in either a loss of material or degradation of mobility below two, this value, and the associated move, replaces the previous best value.

When all possible moves into the center control squares and the supporting squares have been proposed, the best value contains the program's choice. This choice is the move resulting in the maximum center control value. If no moves meet the criteria, the next goal is investigated.

4. Mobility

Moves generated by the generator for MOBILITY are defensive in nature. All possible moves are considered. The final choice is that move which improves mobility the most, without loss of material. The generator will accept exchanges to improve mobility, but sacrifices for mobility are only accepted by the sacrifice move generator.

5. <u>Capture</u>

The move generator for the CAPTURE goal considers all legal moves. After each move is proposed and the resulting board is reduced to a static position, the opponent's mobility is measured.

If a proposed move creates a threat to an enemy piece, this move is analyzed further. The program assumes that the jump is performed and then investigates all possible replies that the opponent can make. If this look-ahead indicates that the opponent cannot gain material, the move is accepted. As soon as an acceptable threatening move is found, it is selected. If no threatening move which meets the criteria is discovered, the move which reduces the opponent's mobility the most, without degrading material condition, is selected.

6. <u>Exchange</u>

The move generator the EXCHANGE goal searches for a move which will result in an even exchange of material. The generator considers each legal move. The first move which results in a static position with less men on the board and the same material balance is selected.

7. <u>Get-king</u>

.The move generator for the GET-KING goal first attempts to move a man directly into kings row. If no man is in position for such a move, the generator attempts to determine the opponent's weak side of the board. Squares on the board are assigned values corresponding to the value derived in defense of kings row by occupying the squares.[3] The

[3] Appendix C illustrates the numbering scheme for computing the opponent's weak side.

values of all the squares occupied by the opponent are summed for each side of the board. The side with the lowest defensive value is considered to be the opponent's weak side. The generator assumes that the program is most likely to penetrate the opponent's defense on this side.

The generator first proposes advancing the man already closest to the kings row on the weak side. The move is proposed and the board is reduced to a static position. If the man is still on the square it moved to, and material balance is unchanged, the move is accepted. If the move is not acceptable, the next closest man on the weak side is considered for advancement.

Considering moves in this order allows the program to continue with some sense of direction toward getting a king. Often the same man is advanced several moves in a row. If conditions on the weak side change, however, and another man is in a better position to reach kings row, the generator will select the better move. One draw-back to successively moving the same man is the possibility of isolating him from defensive support. This situation has not occurred frequently in play. Moves which place the advancing man on the edge of the board (squares 5, 13, 14, 22, 23, and 31) are more likely to cause the man to be trapped. These squares are only moved into when moves nearer the centerline of the board are not acceptable.

The first move to meet the criteria is selected.

8. Safe-move and Sacrifice

If, after all the goals have been processed, the program cannot find a move which meets the requirements of one of the goals, the SAFE-MOVE move generator selects one of the safe moves available to the program. Safe moves are actually detected previously by the routine which computes mobility. This routine saves one of the safe moves for the move generator for the SAFE-MOVE goal.

In the event that no safe moves exist, the SACRIFICE move generator will select a move to sacrifice a man. The sacrifice move selected is the move which results in a dead position containing at least one safe move for the program and results in the smallest loss in material. Sacrifices which do not result in safe moves at the dead position are rejected. The program resigns if this generator fails to produce a move.

9. Summary

In summary, the final choice of a move by the move generator is either the move producing the most improvement in the associated goal, or the first to produce improvement. The center control and mobility move generators are also sensitive to moves which produce a gain in material for the program. Time did not permit including this feature in the other move generators.

If a move is proposed to improve center control or mobility, and a material gain results at the dead position, the move is selected regardless of the effect on center control or mobility. Likewise, if a move results in a loss of material, except in an exchange, the move is rejected unless the SACRIFICE move generator is executing.

At one point in the development of the move generators, they used one ply look-ahead to analyze each of the moves proposed. Because of the excessive time involved in this analysis, only the CAPTURE move generator retains this feature. Even in this generator, only moves which place an enemy piece in danger are so analyzed.

D. PROGRAMMING

All programming associated with this work has been done using the LISP 1.5 Programming Language [Refs. 7 and 8]. LISP 1.5 is a list processing system based on the formal LISP language, using an interpreter for evaluating LISP expressions. The language is fully recursive and particularly well suited to manipulation of strings and lists.

The first step in programming was to code definitions in LISP for various words and phrases associated with checkers. Basic machinery was developed to change the board to reflect moves and jumps, and to measure mobility, center control, and material condition. The move generation phase was then coded, using the definitions of the goals to determine criteria for acceptance of a move, and the previously coded definitions associated with each goal.

The coded definitions are similar to the chess vocabulary of Newell, Simon, and Shaw, in that a function, called with the proper color and board, will return information relevant to goal and move generation. For example, one function, called ALL-MOVES, returns all legal moves a player has available.

With this vocabulary of functions, the goal and move generation functions operate on a higher language level, by doing their analysis and evaluation using these definitions.

Near the end of the project, a new version of the LISP programming system for IBM S/360 Operating Systems was received from IBM [Ref. 9]. The new version compiled machine code to perform the manipulations and was compatible with the time-sharing system installed at the Naval Postgraduate School.

The overall supervisory routines were altered to make the checker-playing program interactive. Some changes were also required in the remaining sections of the program to insure compatibility. With these changes effected, the program was then used interactively for play and program de-bugging.

Considerable time and effort was expended to implement the program on the time-sharing system. Some difficulty was experienced late in the work which precluded a detailed analysis and comparison of the processing performance of the program on the two systems. It was expected that the time-sharing system, using compiled code for execution, would require considerably less processing time per move than the interpreter system.

Because of the time required to perform needed changes to the program to make it compatible with the time-sharing system version of LISP, no data is available for comparison at this time.

E. PROGRAM PERFORMANCE

The program has played five games from beginning to end. The games show that improvement has been made in several areas of play. The processing times and free storage requirements for the program have been reduced, and the program is able to cope with many more situations than the original program could.

The original program played three games. These games, although very weak in the middle and end game, were helpful in bringing to light areas that needed improvement. The beginning was the best part of the game played by the program, but this was over-directed toward the CENTER-CONTROL goal, which limited the aggressiveness of the program.

The original player required between 30 and 90 seconds of processing time for a single move in the beginning game. In the middle game, moves frequently would require over four minutes of processing time; especially when the program could not generate moves for the first or second goals on the goal list. Processing times in the end game were near two minutes, but in many cases the program could not find an acceptable move for any of the goals, and conceded when the situation was not hopeless. At that time there were no generators for exchanges, safe moves, or sacrifices. These processing times were achieved using 34996 words of free storage for LISP.

Two full games have been played using the present program and the batch (interpreter) system. Processing times are now between 20 and 60 seconds for each move in the beginning and end game situations. Averages for the middle game are slightly higher, but seldom does a move require more than 90 seconss of processing. These times were achieved using 24446 words of free storage.

This improvement in the processing times is a result of more efficient coding and more reasonable ordering of goals in the goal generation phase. Near the very end of the game - say with only three or four men left - processing times may rise to near three minutes. This problem can be reduced by elimination of some goals during this stage of the game. Appendix A contains sections of games that were played, with comments as to what move generator selected the move, and moves that may have been better.

The program plays best in the beginning game. Exchanges are used for accomplishing goals more than they are in published games by experts , but the exchanges result in the most improvement in the associated board features.

The middle game lacks the direction necessary to press on toward strategic gains, because the program does not remember previous goals or moves. Each goal generation - move generation - analysis - choice cycle is performed with only the present board position as context.

The end game is weak - especially when the program has the material advantage and therefore has the potential to win. The primary method the program has to recognize progress toward a win is measurement of the opponent's mobility. A reduction in the opponent's mobility is seen as advancement toward a win. If, as in game four in Appendix A, the program's pieces are separated from his opponent's pieces, such that no move will reduce the opponent's mobility, the capture goal cannot be accomplished. The move selected by the program will not necessarily move a piece closer to the opponent.

Overall, the program can play an interesting game, but cannot push on to a win. More detailed analysis can help in this respect. Addition of a goal to close with enemy pieces if certain circumstances exist could also help the end-game performance. With this goal, in situations such as in game four, the program would be able to approach the opponent and then find moves to restrict his mobility - forcing the win.

VI. CONCLUSION

The checker-playing program described in this paper is based on Shannon's original proposal for playing chess. The program embodies many of the ideas presented by researchers in computer game playing - with the obvious omission of extensive look-ahead. The performance of this program, along with that of previously written game-playing programs, yields information supporting several conclusions about game playing in general and CHECKER1 in particular.

A. SHANNON'S INFLUENCE ON DESIGN

Almost every attempt to write a computer program to play chess or checkers has been patterned after Shannon's original framework. To date, no program has been produced which can play these games well enough to consistently beat excellent human players. However, as workers have implemented Shannon's proposal better and built on other concepts in the literature, program performance has improved. Some programs, such as the Greenblatt Chess Program, have been successful against fair to good players, but none has achieved the success that many people consider possible. This could be because of incomplete implementation of Shannon's proposal or because his design is inadequate to produce high quality play.

B. PERFORMANCE FACTORS

The most successful game-playing programs contain considerable built-in game knowledge. Using stored openings and tricks of the game is one method that program designers have used to improve the performance of their programs.

All of the game playing programs discussed in this paper use some scheme to produce a numerical evaluation of a board position. The evaluation is based on measurements of various relevant features of the board position. If the figure of merit assigned to the board is to be of worth in selecting moves, many features of the board should be considered and the entire spectrum of information derived from the situation should be used. Programs which reduce this information to a single figure, as Samuel does, or keep the information distinct and then allow one feature to totally dominate move selection, as in the NSS Chess Player, may not be making full use of the available information.

Although CHECKER1 does not measure many features of the board, move selection decisions are based on several of the measurements. The program makes use of all the information obtained about the board features it measure during analysis.

An integral part of Shannon's chess playing design is the use of look-ahead. He proposed that the farther the program looked ahead, the better the program could play. Extending the look-ahead deeper into the tree gives the program more assurance that its evaluation of a move is reasonable. However, the value derived from look-ahead is based on the ability the program has to perform good static evaluations at the terminal nodes of the look-ahead. If these board position evaluations are accurate, the look-ahead will actually introduce information not available from the original board position. Without good evaluations, the look-ahead value is greatly reduced.

A program using no look-ahead could be expected to play chess or checkers well if it were able to analyze the current board configuation sufficiently. No look-ahead implies that the program can calculate directly the best move available to the player. This move could be determined by careful analysis of the board position. Plausible moves would be proposed and the resulting board position reduced to a dead position. This board would then be analyzed for move selection. Since the opponent's replies would not be investigated, this procedure would not be true look-ahead.

A program using no look-ahead would require a number of relevant board feature measurements and the ability to recognize the relative importance of each. The program would contain much specific information about the game, including stored moves obtained from the literature of the game.

CHECKER1 only uses look-ahead in particular applications of the move generator for the CAPTURE goal. The look-ahead is used to detect situations which could be recognized using a pattern matching technique. Because this look-ahead can be replaced without any effect on CHECKER1's performance, the program effectively contains no look-ahead. The ability CHECKER1 has to select reasonable moves supports the assertion that checkers can be played without look-ahead.

The program is capable of play that is at least interesting, using a very limited set of board features. Adding new features and additional analysis routines to the goal selection and move selection phases should yield a program with improved performance. Experimentation with this improved program should produce additional insight into the capabilities of a no look-ahead checker-playing program.

APPENDIX A

COMPUTER CHECKER GAMES

Game 1 Basic Checker Program

	BLACK - Checker	RED - Students	COMMENTS
1.	12 - 16	24 - 20	CENTER-CONTROL
2.	16 - 24 Jump	28 - 20 Jump	
3.	8 - 12	32 - 28	Single corner opening.
4.	4 - 8	28 - 24	
5.	11 - 16	26 - 28	CENTER-CONTROL
6.	10 - 15	20 - 10 Jump	Exchange for CENTER-CONTROL
7.	5 - 15 Jump	25 - 21	
8.	16 - 26 Jump	31 - 21 Jump	
9.	12 - 16	21 - 11 Jump	Exchange for CENTER-CONTROL
10.	6 - 16 Jump	30 - 25	
11.	16 - 21	25 - 17 Jump	Exchange for GET-KING
12.	13 - 21 Jump	24 - 19	
13.	7 - 12	19 - 11 Jump	GET-KING: couldn't block.
14.	12 - 16	23 - 19	CENTER-CONTROL
15.	8 - 12	19 - 15	CENTER-CONTROL
16.	16 - 20	35 - 30	GET-KING. Red sets a bridge.
17.	12 - 16	15 - 10	CENTER-CONTROL. (21-26 better)

At this point the game was discontinued. The program could not move men out of danger, was over-directed toward CENTER-CONTROL, and did not see the bridge opening kings row.

Game 2 Basic Checker Program

	BLACK - Checker	RED - Students	COMMENTS
1.	10 - 15	24 - 10	
2.	12 - 16	20 - 10 Jump	Program did not block jump.
3.	5 - 15 Jump	25 - 21	Exchange anyway.

	BLACK - Checker	RED - Students	COMMENTS
4.	16 - 20	29 - 24	
5.	11 - 16	21 - 11 Jump	Exchange for CENTER-CONTROL
6.	6 - 16	24 - 19	
7.	7 - 12	19 - 11	Program did not block.
8.	20 - 24	28 - 20 Jump	
9.	16 - 24 Jump	23 - 18	
10.	13 - 17	19 - 14	
11.	12 - 16	26 - 22	
12.	8 - 12	33 - 29	BLOCK now working.
13.	16 - 21	29 - 19	2 - 7 better move for black.
14.	12 - 16	22 - 2 Jump	Double jump for Red.
15.	2 - 7	19 - 15	From GET-KING, not KIL
16.	16 - 20	30 - 25	Red sets Black up.
17.	21 - 25 Jump	34 - 16 Jump	Two-for-one shot for Red.
18.	1 - 5	31 - 26	

Game 3 Basic Checker Program

	BLACK - Checker	RED - Students	COMMENTS
1.	10 - 15	24 - 20	
2.	5 - 10	38 - 24	Black blocks jump.
3.	12 - 16	20 - 12 Jump	Exchange for CENTER-CONTROL.
4.	8 - 16 Jump	26 - 22	
5.	13 - 17	22 - 12 Jump	
6.	7 - 17 Jump	30 - 12 Jump	Two-for-one shot for RED.
7.	10 - 15	24 - 20	CENTER-CONTROL.
8.	15 - 25 Jump	29 - 13 Jump	Two-for-one shot for RED.
9.	11 - 15	31 - 26	
10.	6 - 11	23 - 29	
11.	15 - 20	23 - 19	
12.	20 - 25	32 - 28	BLACK moving for GET-KING.
13.	2 - 6	12 - 7	Open bridge for RED.
14.	11 - 15	19 - 11 Jump	
15.	6 - 26	7 - 2	Two-for-one shot for BLACK. RED gets a king.
16.	26 - 31	24 - 20	
17.	1 - 5	20 - 15	
18.	4 - 8	34 - 30	

	BLACK - Checker	RED - Students	COMMENTS
19.	8 - 12	30 - 20	BLACK found only safe move.
20.	12 - 17	20 - 16	
21.	17 - 22	16 - 11	
22.	22 - 26		

The game was finished at this point. RED had a good material advantage, but BLACK did find a two-for-one shot at move 15. BLACK's moves helped RED set up his two two-for-one shots.

Game 4 CHECKER1

	BLACK - CHECKER1	RED - De Ford	COMMENTS
1.	13 - 17	25 - 20	CENTER-CONTROL
2.	10 - 15	20 - 10 Jump	Exchange for CENTER-CONTROL
3.	6 - 14 Jump	24 - 19	
4.	14 - 24 Jump	28 - 20 Jump	
5.	11 - 15	20 - 10 Jump	Exchange for CENTER-CONTROL
6.	5 - 15 Jump	29 - 25	
7.	7 - 11	25 - 21	BLACK moves for GET-KING. RED sets up a two-for-one shot.
8.	17 - 25 Jump	30 - 10 Jump	Two-for-one shot for RED.
9.	1 - 5	10 - 6	BLACK gets KIL.
10.	2 - 10	26 - 21	
11.	11 - 15	32 - 28	CENTER-CONTROL.
12.	15 - 20	34 - 29	BLACK moves for GET-KING.
13.	20 - 25	23 - 19	BLACK moves into bridge.
14.	8 - 18	31 - 26	GET-KING.
15.	12 - 17	26 - 22	GET-KING.
16.	3 - 8	22 - 12 Jump	BLACK could not block, so caused an exchange.
17.	8 - 26 Jump	29 - 21 Jump	BLACK gets two men. Could not save other man.
18.	26 - 31	33 - 29	BLACK gets trapped in 31.

	BLACK - CHECKER1	RED - DE Ford	COMMENTS
19.	4 - 8	28 - 24	GET-KING.
20.	8 - 12	24 - 20	BLACK moves for MOBILITY.
21.	12 - 17	21 - 16	BLACK gets trapped in 17. Program changed to avoid edge.
22.	17 - 22	16 - 11	
23.	10 - 15	20 - 10 Jump	BLACK sees two-for-one shot.
24.	5 - 23 Jump	11 - 6	Two-for-one. RED heads for King.
25.	23 - 28	6 - 2	BLACK moves for GET-KING. RED gets a king.
26.	28 - 33	29 - 25	BLACK gets a king.
27.	22 - 26	25 - 20	CENTER-CONTROL.
28.	33 - 29	20 - 16	MOBILITY.
29.	29 - 24	16 - 11	CENTER-CONTROL.
30.	13 - 17	11 - 7	GET-KING.
31.	17 - 21	7 - 3	RED gets a king.
32.	21 - 25	3 - 8	GET-KING.
33.	25 - 29	2 - 6	
34.	29 - 34	8 - 12	BLACK gets a king.
35.	24 - 20	6 - 1	CENTER-CONTROL.
36.	34 - 29	12 - 17	MOBILITY.
37.	20 - 16	17 - 22	BLACK moves to CAPTURE.
38.	26 - 30	35 - 25 Jump	Moves to exchange kings since not behind in material. (29 - 34 is better)
39.	29 - 21 Jump	1 - 6	
40.	31 - 35	6 1 10	BLACK gets a king.
41.	21 - 26	22 - 30 Jump	BLACK moves to exchange kings.
42.	35 - 25 Jump	10 - 5	

The game ended when the program resorted to any safe move. Because no move would restrict RED's mobility, BLACK could not satisfy the CAPTURE goal. This game was played for testing and experimenting purposes. RED did not play to win in all situations.

Game 5 - CHECKER_1

	BLACK - CHECKER_1	RED - De Ford	COMMENTS
1.	10 - 15	26 - 21	
2.	11 - 16	21 - 11 Jump	Exchange for CENTER-CONTROL.
3.	6 - 16 Jump	24 - 19	
4.	7 - 11	19 - 14	BLACK blocked jump.
5.	5 - 10	14 - 6	BLACK moves to exchange.
6.	2 - 10 Jump	29 - 24	
7.	10 - 14	24 - 20	RED sets up a two-for-one shot.
8.	16 - 24 Jump	28 - 10 Jump	RED gets two-for-one shot.
9.	11 - 15	33 - 28	CENTER-CONTROL.
10.	13 - 17	31 - 26	GET-KING.
11.	17 - 22	26 - 21	GET-KING.
12.	14 - 19	24 - 20	GET-KING on opposite side.
13.	15 - 25 Jump	30 - 20 Jump	RED forces exchange. BLACK man on 19 in danger.
14.	19 - 24	21 - 16	BLACK moves man out of danger.
15.	22 - 26	16 - 11	GET-KING.
16.	12 - 16	20 - 12 Jump	Exchange for CENTER-CONTROL.
17.	8 - 16	28 - 12 Jump	Blunder by BLACK gives RED two men.
18.	3 - 8	12 - 7	SAFE-MOVE. RED goes for king.
19.	8 - 12	7 - 3	RED gets a king.

The game was discontinued. RED had a three-man material advantage. A flaw in the static analysis routine caused the program to not recognize the danger of move 17.

APPENDIX B

CHECKER1 ON TIME-SHARING SYSTEM

Although no complete games were played using the time-sharing system, some portions of games were played. Reference 9 gives detailed information on the use of LISP 1.5 on CP/CMS. The following is an example of one terminal session after CHECKER1 had been compiled and stored as part of the LISP system.

```
login 1875g13          (LISP 1.5 requires 512 K virtual memory)
ENTER PASSWORD:
npg
ENTER 4-DIGIT PROJECT NUMBER FOLLOWED BY 4-CHARACTER COST
CENTER CODE:
0530cs04
SHUTDOWN TIME FOR CP IS 1600 SHARP..HAVE A NICE DAY..DUFFY
READY AT 12.09.16 ON 11/19/71
CP
.i cms
CMS..VERSION 01/21/71

cp link 1875p 191 192 w    (Link to private file)
ENTER PASSWORD:
checker
R; T=0.03/0.19 12.09.51

login 192
** 192 REPLACES P (191) **
R; T-0.02/0.14 12.09.57

lisp117 lisp117        (Loads and executes LISP system)
12.10.0.7 CHARDEF T
12.10.09 CHARDEF B
12.10.11 LINEND
12.10.12 LOAD LISP117 (CLEAR)
12.10.25 DEBUG
DEBUG   ENTERED...
```

```
12.10.29 START LISPHOT LISP117
EXECUTION BEGINS...
LISP VERSION 117-0
CORE IMAGE:   LISP117   SOSTAP     P1
VALUE =   3

play()
PLAY NIL

WHAT_BOARD_WOULD_YOU_LIKE_TO_START_WITH?
SAY_INITIAL_IF_WE_ARE_TO_START_A_FRESH_GAME
SAY_DATA_IF_YOU_WANT_TO_START_WITH_A_PARTICULAR_BOARD
INITIAL
HERE_IS_THE_BOARD
((CURRENT_BOARD . 0)  (1 . 1) (2 . 1)  (3 . 1)  (4 . 1)  (5 . 1)  (6 . 1)
(7 . 1)  (8 . 1)  (10 .  1)  (11 . 1)  (12 . 1)  (13 . 1)  (14 . 0)  (15 . 0)
(16 . 0)  (17 . 0)  (19 . 0)  (20 . 0)  (21 . 0)  (22 . 0)  (23 . -1)  (24 . -1)
(25 . -1)  (26 . -1)  (28 . -1)  (29 . -1)  (30 . -1)  (31 . -1)  (32 . -1)
(33 . -1)  (34 . -1)  (35 . -1))

WHICH_COLOR_WOULD_YOU_LIKE_ME_TO_PLAY?
BLACK
HERE_IS_THE_BOARD
((CURRENT_BOARD . 0)  (1 . 1)  (2 . 1)  (3 . 1)  (4 . 1)  (5 . 1)  (6 . 1)
(7 . 1)  (8 . 1)  (10 . 1)  (11 . 1)  (12 . 1)  (13 . 1)  (14 . 0)  (15 . 0)
(16 . 0)  (17 . 0)  (19 . 0)  (20 . 0)  (21 . 0)  (22 . 0)  (23 . -1)  (24 . -1)
(25 . -1)  (26 . -1)  (28  . -1)  (29 . -1)  (30 . -1)  (31 . -1)  (32 . -1)
(33 . -1)  (34 . -1)  (35 . -1))
MY GOAL_LIST_IS_:
(SHOT KIL CENTER_CONTROL EXCHANGE CAPTURE GET_KING
MOBILITY)

I_EVALUATE_THE_BOARD:
(0 0 7 0)

MY_MOVE_IS:
(MOVEPIECE 1 10 . 15)

((CURRENT_BOARD  . 1)  (1 . 1)  (2 . 1)  (3 . 1)  (4 . 1)  (5 . 1)  (6 . 1)
(7 . 1) (8 . 1)  (10 . 0)  (11 . 1)  (12 . 1)  (13 . 1)  (14 . 0)  (15 . 1)
(16 . 0)  (17 . 0)  (19 . 0)  (20 . 0)  (21 . 0)  (22 . 0)  (23 . -1)  (24 . -1)
(25 . -1)  (26 . -1)  (28 . -1)  (29 . -1)  (30 . -1)  (31 . -1)  (32 . -1)
(33 . -1)  (34 . -1)  (35 . -1))
IT'S_YOUR_MOVE
```

```
(MOVE 26 21)
HERE_IS_THE_BOARD
((CURRENT_BOARD . 2)  (1 . 1)  (2 . 1)  (3 . 1)  (4 . 1)  (5 . 1)
(6 . 1)  (7 . 1)  (8 . 1)  (10 . 0)  (11 . 1)  (12 . 1)  (13 . 1)  (14 . 0)
(15 . 1)  (16 . 0)  (17 . 0)  (19 . 0)  (20 . 0)  (21 . -1)  (22 . 0)  (23 . -1)
(24 . -1)  (25 . -1)  (26 . 0)  (28 . -1)  (29 . -1)  (30 . -1)  (31 . -1)
(32 . -1)  (33 . -1)  (34 . -1)  (35 . -1))
MY_GOAL_LIST_IS_:
(SHOT KIL CENTER_CONTROL EXCHANGE CAPTURE GET_KING MOBILITY)

I_EVALUATE_THE_BOARD:
(0 0 4 0)

MY_MOVE_IS:
(MOVEPIECE 1 11 . 16)

((CURRENT_BOARD . 3)  (1 . 1)  (2 . 1)  (3 . 1)  (4 . 1)  (5 . 1)
(6 . 1)  (7 . 1)  (8 . 1)  (10 . 0)  (11 . 0)  (12 . 1)  (13 . 1)  (14 . 0)
(15 . 1)  (16 . 1)  (17 . 0)  (19 . 0)  (20 . 0)  (21 . -1)  (22 . 0)  (23 . -1)
(24 . -1)  (25 . -1)  (26 . 0)  (28 . -1)  (29 . -1)  (30 . -1)  (31 . -1)
(32 . -1)  (33 . -1)  (34 . -1)  (35 . -1))
IT'S_YOUR_MOVE

(I QUIT)

PRESENT_BOARD

((CURRENT_BOARD . 3)  (1 . 1)  (2 . 1)  (3 . 1)  (4 . 1)  (5 . 1)  (6 . 1)
(7 . 1)  (8 . 1)  (10 . 0)  (11 . 0)  (12 . 1)  (13 . 1)  (14 . 0)  (15 . 1)
(16 . 1)  (17 . 0)  (19 . 0)  (20 . 0)  (21 . -1)  (22 . 0)  (23 . -1)  (24 . -1)
(25 . -1)  (26 . 0)  (28 . -1)  (29 . -1)  (30 . -1)  (31 . -1)  (32 . -1)
(33 . -1)  (34 . -1)  (35 . -1))
```

APPENDIX C

COMPUTATION OF CENTER CONTROL AND WEAK SIDE

The value of center control is an integer which indicates the relative control of the center for the two players. The value is computed by multiplying the value of each piece occupying one of the squares 10, 11, 12, 15, 16, 17, 19, 20, 21, 24, 25, 26 by the center control value of the square, and summing the products. Figure 8 shows the center control values of the squares. The values assigned to the pieces on the checkerboard are:

PIECE	VALUE
BLACK MAN	1
BLACK KING	2
RED MAN	-1
RED KING	-2

The sign of the center control value indicates which player has the advantage (minus for red, plus for black). The magnitude of the value indicates the magnitude of the advantage. If the center control value is zero, neither player has a center control advantage.

As an example computation, consider a board position with black men on squares 5, 10, 12, 15, 17, and 21, red men on squares 20, 25, 26, 29, and 31. The men on squares 5, 29, and 31 do not contribute to center control. The value of center control is computed as follows:

SQUARE	SQUARE VALUE	PIECE VALUE	PRODUCT
10	1	1	1
11	1	0	0
12	1	1	1
15	4	1	4
16	4	0	0
17	1	1	1
19	1	0	0
20	4	-1	-4
21	4	1	4
24	1	0	0
25	1	-1	-1
26	1	-1	-1
	CENTER CONTROL	=	5

When the move generator for GET-KING cannot move a man directly into kings row, it attempts to determine the best move to make to penetrate the opponent's kings row defense. The weak side is determined by summing the defensive values assigned to each square that the opponent occupies for the two board quadrants adjacent to kings row. The side with the minimum value is the weak side.

Figure 9 shows the values assigned to each square. These values were derived by analysis of various board configurations and by consulting the literature on the game of checkers.

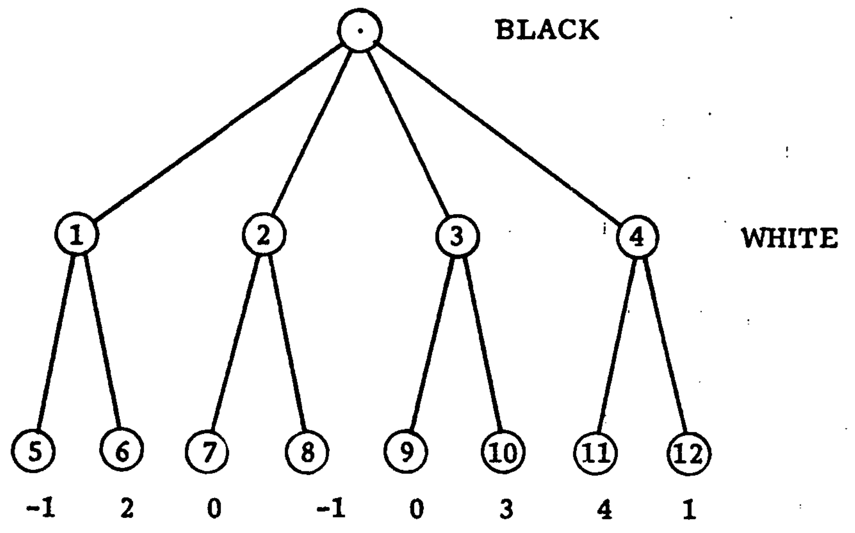


Figure 1. The Move Tree and Minimaxing.

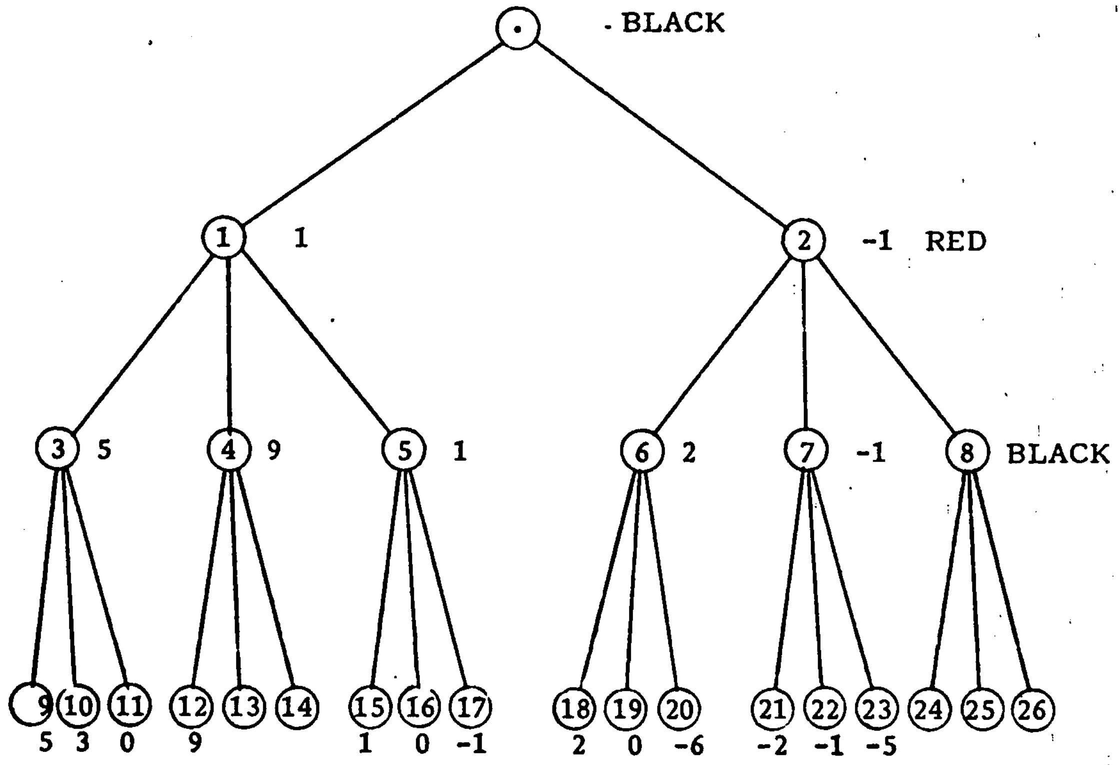


Figure 2. Alpha-Beta Tree Pruning

BLACK

▨	1	▨	2	▨	3	▨	4
5	▨	6	▨	7	▨	8	▨
▨	9	▨	10	▨	11	▨	12
13	▨	14	▨	15	▨	16	▨
▨	17	▨	18	▨	19	▨	20
21	▨	22	▨	23	▨	24	▨
▨	25	▨	26	▨	27	▨	28
29	▨	30	▨	31	▨	32	▨

RED

Figure 3. Standard Checkerboard Numbering System

BLACK

▨	1	▨	2	▨	3	▨	4
5	▨	6	▨	7	▨	8	▨
▨	10	▨	11	▨	12	▨	13
14	▨	15	▨	16	▨	17	▨
▨	19	▨	20	▨	21	▨	22
23	▨	24	▨	25	▨	26	▨
▨	28	▨	29	▨	30	▨	31
32	▨	33	▨	34	▨	35	▨

RED

Figure 4. Modified Checkerboard Numbering System.

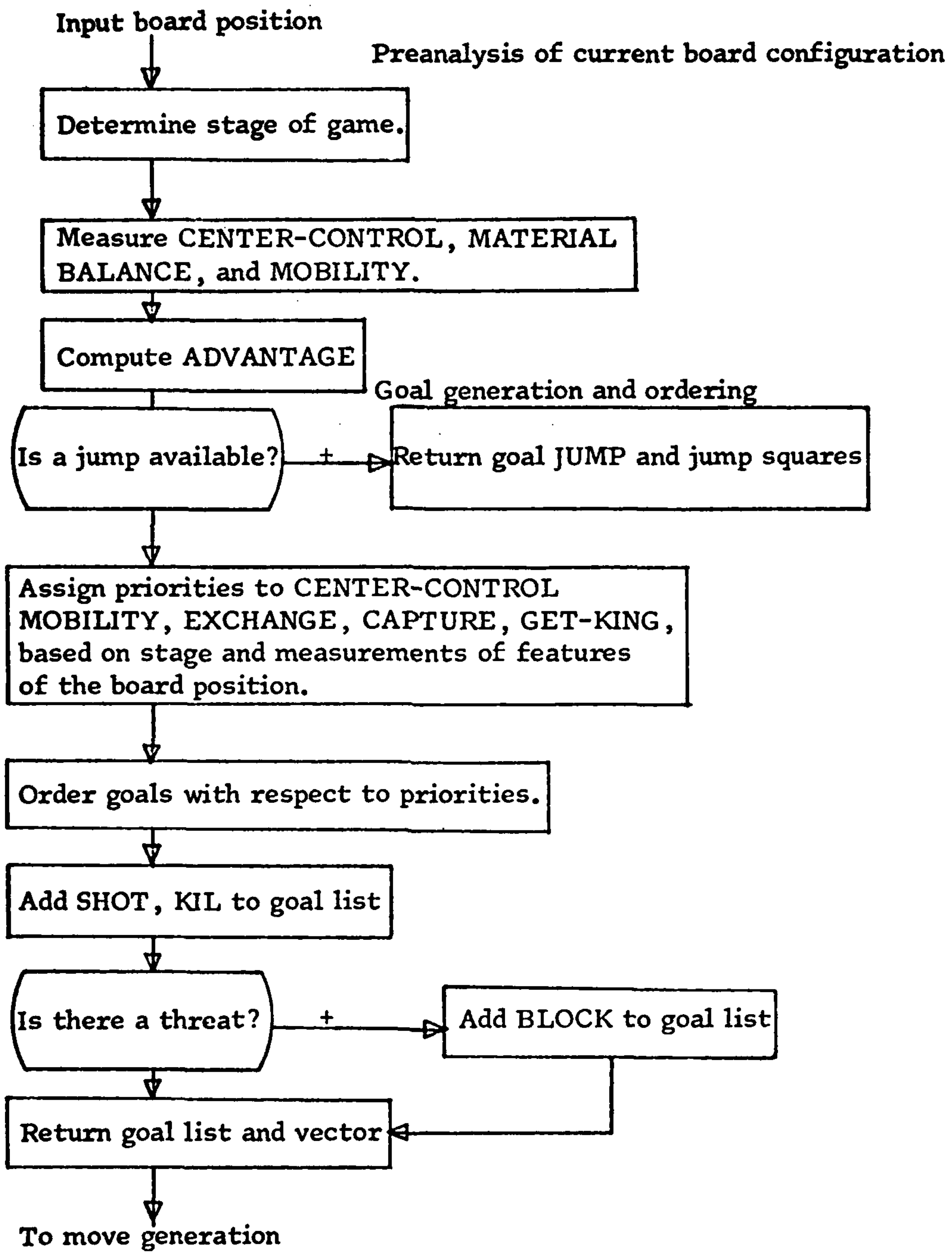
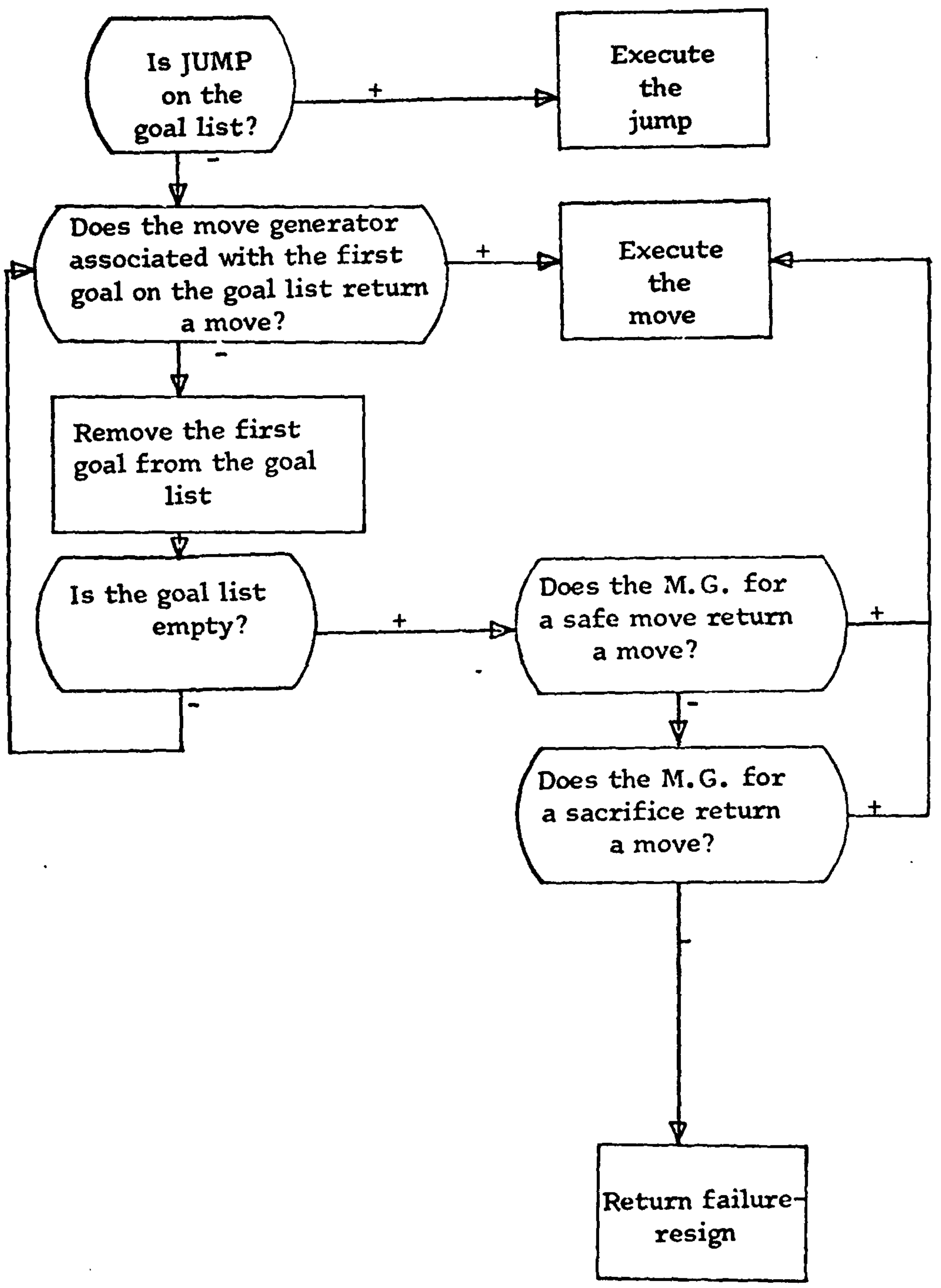

Input board position
Preanalysis of current board configuration
Determine stage of game.
Measure CENTER-CONTROL, MATERIAL BALANCE, and MOBILITY.
Compute ADVANTAGE
Goal generation and ordering
Is a jump available?
+
Return goal JUMP and jump squares
Assign priorities to CENTER-CONTROL MOBILITY, EXCHANGE, CAPTURE, GET-KING, based on stage and measurements of features of the board position.
Order goals with respect to priorities.
Add SHOT, KIL to goal list
Is there a threat?
+
Add BLOCK to goal list
Return goal list and vector
To move generation

Figure 5. Pre-analysis and Goal Generation.

Input: Goal list and Vector



Is JUMP on the goal list?
+
Execute the jump
-
Does the move generator associated with the first goal on the goal list return a move?
+
Execute the move
-
Remove the first goal from the goal list
Is the goal list empty?
+
Does the M.G. for a safe move return a move?
+
-
-
Does the M.G. for a sacrifice return a move?
+
-
Return failure-resign

Figure 6. Move Generation

GOAL	MOVES CONSIDERED	CHOICE CRITERION
BLOCK	All moves into block square.	First with no threat.
	All moves for threatened man.	First with no threat.
	All moves into squares adjacent to the block square.	First to cause exchange.
SHOT		Pattern recognition.
KIL		Pattern recognition.
CENTER-CONTROL	All moves into Center Control and Supporting squares.	Move resulting in best center control value.
MOBILITY	All moves	Move resulting in best mobility.
GET-KING	All moves into kings row.	First to get a king.
	All moves on the opponent's weak side.	Move putting man closest to kings row
	All moves on the opponent's strong side.	Move putting man closest to kings row.
CAPTURE	All moves.	First placing enemy in danger.
	All moves.	Move causing minimum enemy mobility.
EXCHANGE	All moves.	First causing exchange.
SAFE-MOVE	All moves.	First safe move.
SACRIFICE	All moves.	First sacrifice causing mobility greater than zero.

Figure 7. Comparison of Move Generators.

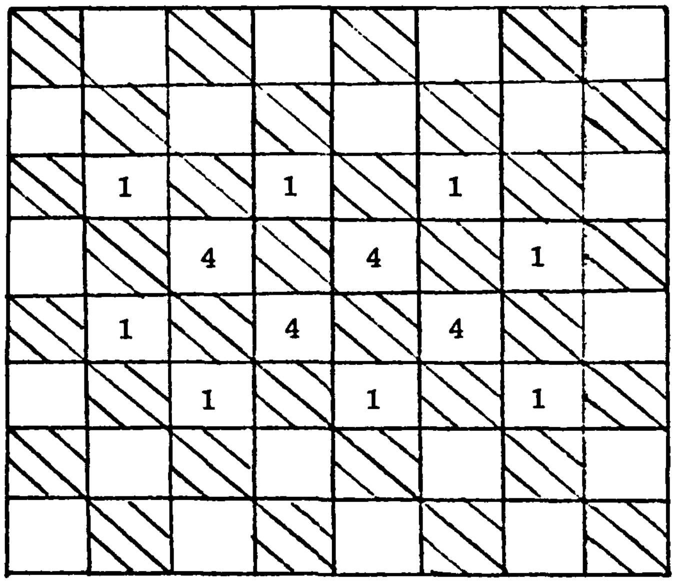


Figure 8. Center Control Square Values.

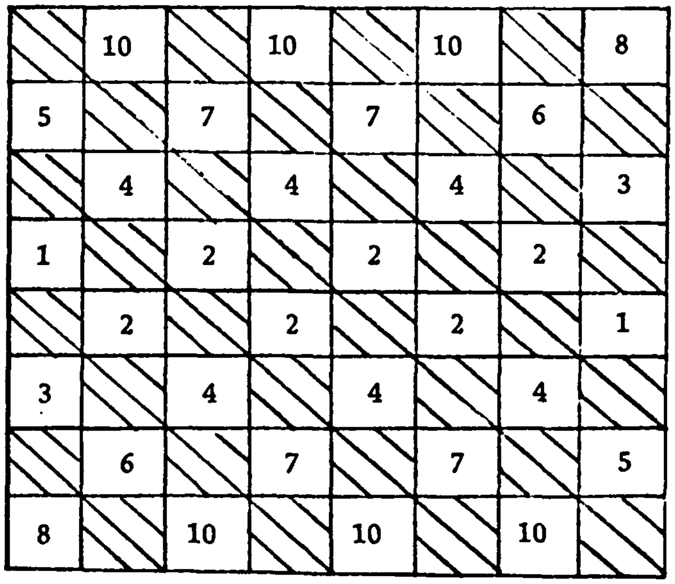


Figure 9. Defensive Value Assignments.

LIST OF REFERENCES

1. Shannon, C. E., "Programming a Digital Computer for Playing Chess," Philosophy Magazine, v. 41, p. 356-375, March 1950.

2. Turing, A. M., Faster Than Thought, p. 288-295, Putman, 1953.

3. Bernstein, A., Roberts, M. DE. V., Arbuekle, T., and Belsky, M. A., "A Chess Playing Program For The IBM-704 Computer," Proc. 1958 Western Joint Computer Conference, p. 157-159.

4. Newell, A., Simon, H., and Shaw, J. C., "Chess Playing Programs And The Problem of Complexity," IBM Journal of Research and Development, v. 2, p. 320-355, October 1958.

5. Greenblatt, R. D., Eastlake, D. E. III, and Crocker, S. D., "The Greenblatt Chess Program," Fall Joint Computer Conference, 1967, p. 801, 810.

6. Samuel, A. L., "Some Studies In Machine Learning Using The Game of Checkers," Computers and Thought, p. 71-105, 1963.

7. McCarthy, J., and others, LISP 1.5 Programmer's Manual, The MIT Press, 1962.

8. Weissman, C., (LISP 1.5 Primer(by(Clark Weissman))), Dickenson, 1968.

9. Blair, F. W., and others, Design and Development Document for LISP on Several S/360 Operating Systems, Yorktown Heights, New York.